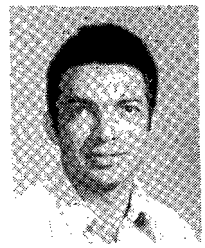
[14] H. J. Payne and W. A. Thompson, "Traffic assignment on transportation networks with capacity constraints and queueing," in *Proc. 47th Nat. ORSA Meeting*, 1975.
[15] R. G. Miller, "The jackknife–A review," *Biometrica*, vol. 61, no. 1, pp. 1-15, 1974.

★



Adrian Segall (S'71-M'74) was born in Bucharest, Romania, on March 15, 1944. He received the B.Sc. and M.Sc. degrees from the Technion–Israel Institute of Technology, Haifa, in 1965 and 1971, respectively, both in electrical engineering, and the Ph.D. degree from Stanford University, Stanford, CA, in 1973 also in electrical engineering, with a minor in statistics.

From 1965 to 1968 he served in the Israel Defense Army as an Electronics Engineer. From 1968 to 1971 he was a Research Engineer in the Scientific Department, Israel Ministry of Defence. From 1971 to 1973 he worked on his Ph.D. dissertation at Stanford University, where he held positions of Research Assistant and Acting Instructor in Electrical Engineering. From 1973 to 1974 he was a Research Engineer at Systems Control, Inc., Palo Alto, CA, and a Lecturer in Electrical Engineering at Stanford University. In 1974 he joined the faculty of the Massachusetts Institute of Technology, Cambridge, where he was an Assistant Professor of Electrical Engineering and Computer Science and a consultant to Codex Corporation. He is now with the Department of Electrical Engineering, Technion IIT, Haifa, Israel. His research interests are in applications of the theory of stochastic processes and optimization to computer communication networks, estimation, detection, and automatic control.

Throughput in the ARPANET–Protocols and Measurement

LEONARD KLEINROCK, FELLOW, IEEE, AND HOLGER OPDERBECK, MEMBER, IEEE

Abstract–**The speed at which large files can travel across a computer network is an important performance measure of that network. In this paper we examine the achievable sustained throughput in the ARPANET. Our point of departure is to describe the procedures used for controlling the flow of long messages (multipacket messages) and to identify the limitations that these procedures place on the throughput. We then present the quantitative results of experiments which measured the maximum throughput as a function of topological distance in the ARPANET. We observed a throughput of approximately 38 kbit/s at short distances. This throughput falls off at longer distances in a fashion which depends upon which particular version of the flow control procedure is in use; for example, at a distance of 9 hops, an October 1974 measurement gave 30 kbit/s, whereas a May 1975 experiment gave 27 kbit/s. The two different flow control procedures for these experiments are described, and the sources of throughput degradation at longer distances are identified, a major cause being due to a poor movement of critical limiting resources around in the network (this we call "phasing"). We conclude that flow control is a tricky business, but in spite of this, the ARPANET throughput is respectably high.**

I. INTRODUCTION

THE ARPANET, which was the world's first large-scale experimental packet-switching network, needs little introduction; it has been amply documented (see, for example, [5] and the extensive references therein). Our interest in this paper is to describe the message-handling protocols and some experimental results for the achievable throughput across the ARPANET. These experiments were conducted at the UCLA Network Measurement Center (NMC) and show that the network can support roughly 38 kbit/sec between HOST computers which are a few hops apart; for more distant HOST pairs, the throughput falls off to a level dependent upon the particular version of message processing used, as discussed in detail below.

An earlier NMC experiment reported upon the behavior of actual user traffic in the ARPANET (and also described the NMC itself) [4]. More recent NMC experiments identified, explained, and solved some deadlock and throughput-degradation phenomena in the ARPANET [11] and also measured the effect of network protocols and control messages on line overhead [4]. The experiments reported upon herein consisted of throughput measurements of UCLA-generated traffic (using our PDP 11/45 HOST in a dedicated mode) which was sent through the ARPANET to "fake" HOST's at various topological distances (hops) from UCLA. Each experiment ran for 10 min during which time full (8-packet) multipacket traffic was pumped into the ARPANET as fast as the network would permit. Both throughput (from the UCLA HOST to the destination HOST) and delay (as seen by the UCLA HOST) were measured, along with some other statistics described below.

This paper is organized as follows. We describe the message-handling procedure for multipacket messages in Section II, identify the limitations this procedure imposes on the throughput in Section III, and then quantitatively report upon the October 1974 throughput experiments in Section IV. The issue of looping in the adaptive routing procedure and its erratic effect on throughput is discussed in Section V. Some

Manuscript received May 11, 1976; revised July 22, 1976. This paper has been presented at the Fourth Data Communications Symposium, Quebec City, P.Q., Canada, October 7-9, 1975. This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC-15-73-C-0368.

L. Kleinrock is with the Department of Computer Science, University of California, Los Angeles, CA 90024.

H. Opderbeck was with the Department of Computer Science, University of California, Los Angeles, CA 90024. He is now with the Telenet Corporation, Washington, DC.

recent changes to the message-processing procedure are described in Section VI, and in Section VII we describe some of its faults, their correction, and the experimentally achieved throughput as of May 1975, using this new procedure.

II. HANDLING OF MULTIPACKET MESSAGES

In this section, we describe the details for handling multipacket messages in the ARPANET as of October 1974[1]; it was at this time that the initial set of throughput experiments reported here was conducted. This discussion will permit us to identify throughput limitations and to discuss system bottlenecks.

We are interested in the transmission of a long data stream which the ARPANET accepts as a sequence of messages (each with a maximum length of 8063 data bits). Each such message in this sequence will be a "multipacket" message (a multipacket message is one consisting of more than one 1008-bit packet). To describe the sequence of events in handling each multipacket message we refer to Fig. 1 (which gives the details for a data stream requiring only *one* full multipacket message for simplicity). A message is treated as a multipacket message if the HOST-IMP interface has not received an end-of-message indication after the input of the first packet is completed (shown as point *a* in Fig. 1). At this time, transmission of the remaining packets of this message from the HOST to the IMP is temporarily halted until the message acquires some network resources as we now describe. First, the multipacket message must acquire a message number (from the IMP) which is used for message sequencing (point *b*); all messages originating at this IMP and heading to the same destination IMP share a common number space. Next, an entry in the *pending leader table* (PLT) must be obtained as shown at point *c*. The PLT contains a copy of the leader of all multipacket messages that are currently being handled by the source IMP. Among other things, the function of the PLT is to construct the packet headers for the successive packets of the multipacket message. Such an entry is deleted and released when the RFNM (the end-to-end acknowledgment whose acronym comes from "ready-for-next-message") is received from the destination IMP. The PLT is shared by messages from all HOST's attached to the same IMP and used for all possible destinations.

After the PLT entry has been obtained by the multipacket message, a table is interrogated to find out whether there are eight reassembly buffers reserved for this source IMP at the desired destination IMP. If this is not the case, a control message REQALL (request for allocation) is generated and sent from the source IMP (also shown at point *c*) to the destination IMP which requests an allocation of these buffers. The protocol is such that this REQALL steals the acquired message number and the PLT entry for its own use at this time. This request is honored by the destination IMP as soon as it has eight buffers available (point *d*). To report this fact a subnet control message ALL (allocate) is returned to the source IMP, thus delivering the 8-buffer allocation. Since the previously acquired

[1] This is the message-handling procedure referred to as "version 2" in [5].

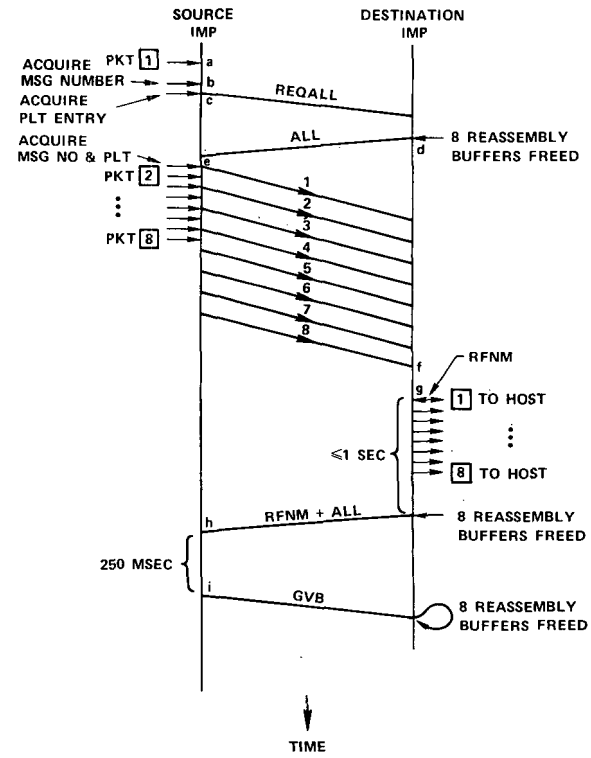


Fig. 1. The sequence of events for one multipacket transmission.

message number and PLT entry have been used, a new message number and a new PLT entry will have to be obtained for the multipacket message itself. (Had 8 reassembly buffers been reserved in the first place, this would have shown at the source IMP by the presence of an unassigned ALL and the steps from *c* to *e* would not have occurred). Only when all these events have taken place can the first packet begin its journey to the destination IMP and can the input of the remaining packets be initiated, as shown at point *e*.

When all packets of the multipacket message have been received by the destination IMP (point *f*), the message is put on the IMP-to-HOST output queue. After the transmission of the first packet to the HOST (point *g*), the RFNM for this message is generated at the destination IMP (also point *g*) to be returned to the source IMP. This RFNM prefers to carry a "piggy-backed" ALL (an implicit reservation of 8 buffers for the next multipacket message) if the necessary buffer space is available. If not, the RFNM will wait for at most 1 s for this buffer space. In case the necessary 8 reassembly buffers do not become available within this second, the RFNM is then sent without a piggy-backed ALL. (We show the case where the buffers do become available in time and so the ALL returns piggy-backed on the RFNM).

After the reception of the RFNM at the source IMP (point *h*), the message number and the PLT entry for this message are freed and the source HOST is informed of the correct message delivery. In case the RFNM carries a piggy-backed ALL, the allocate counter for the proper destination IMP is incremented. This implicit reservation of buffer space is returned to the destination IMP if some HOST attached to the source IMP does not make use of it within the next 250 ms (shown at point *i*); the cancellation is implemented as a control message

GVB (giveback) which is generated at the source IMP. If, however, the next multipacket message to the same destination IMP is received from any source HOST within 250 ms, this message need only acquire a message number and a PLT entry before it can be sent to the destination IMP, and need not await an ALL.

Thus we see that three separate resources must be obtained by each multipacket message prior to its transmission through the net: a message number, a PLT entry, and an ALL.[2]

III. THROUGHPUT LIMITATIONS

Let us now identify the limitations to the throughput that can be achieved between a pair of HOST's in the ARPANET. First we consider the limitations that are imposed by the hardware. The line capacity represents the most obvious and important throughput limitation. Since a HOST is connected to an IMP via a single 100-kbit/s transmission line, the throughput can never exceed 100-kbit/s. If there is no alternate routing in the subnet, the throughput is further limited by the 50-kbit/s line capacity of the subnet communication channels. (The issue of alternate routing is discussed later.)

The processing bandwidth of the IMP allows for a throughput of about 700 and 850 kbit/s for the 316 and 516 IMP's, respectively [6]. Therefore the IMP's can easily handle several 50-kbit/s lines simultaneously. The processing bandwidth of the HOST computers represents a more serious problem. Severe throughput degradations due to a lack of CPU time have been reported in the past [1], [2], [12]. However, these reports also indicate that the degradations are in many cases caused by inefficient implementations of higher level protocols [4]. Therefore, changes in these implementations have frequently resulted in enormous performance improvements [13]. To avoid throughput degradations due to a CPU-limited HOST computer for our throughput experiments, we used a PDP 11/45 minicomputer at UCLA whose only task was to generate 8-packet messages as fast as the network would accept them.

Let us now discuss what throughput limitations are imposed on the system by the subnet *flow control procedure.* As discussed above, there are two kinds of resources a message must acquire for transmission: buffers and control blocks (specifically message numbers and table entries). Naturally, there is only a finite number of each of these resources available. Moreover, most of the buffers and control blocks must be shared with messages from other HOST's. The lack of any one of the resources can create a bottleneck which limits the throughput for a single HOST. Let us now discuss how many units of each resource are available and comment on the likelihood that it becomes a bottleneck. This discussion refers to the ARPANET as of October 1974.

[2] The procedure just described extracts a price for the implementation of its control functions. This price is paid for in the form of overhead in the packets as they are transmitted over the communication channels, in the packets as they are stored in IMP buffers, in control messages (IMP–IMP, IMP–HOST, HOST–HOST), in measurement and monitoring, etc. We refer the reader to [4] for the effect of this overhead on the line efficiency.

In October 1974, a packet was allowed to enter the source IMP only if that IMP had at least four free buffers available. At that time, the total number of packet buffers in an IMP with and without the very distant HOST (VDH) software was, respectively, 30 and 51. This meant that an interruption of message input due to buffer shortage could occur only in the unlikely event that the source IMP was heavily engaged in handling store-and-forward as well as reassembly traffic.

The next resource the message had to obtain was the message number. There was a limitation of only four message sequence numbers allocated per source IMP-destination IMP pair. This meant that all source HOST's at some source IMP A which communicated with any of the destination HOST's at some destination IMP B shared the same stream of message numbers from IMP A to IMP B. This possible interference between HOST's and the fact that there were only four message numbers which could be used in parallel meant that the message number allocation could become a serious bottleneck in cases where the source and destination IMP were several hops apart. (This was the major reason for the recent change to the message processing procedure which has recently been implemented; see Section VI).

After a message number was obtained, the multipacket message had to acquire one of the PLT entries of which there was a shared pool of six. Since the PLT is shared by all HOST's which are attached to the source IMP and used for all possible destinations, it also represents a potential bottleneck. This bottleneck can easily be removed by increasing the number of entries permitted in the PLT. However, the PLT also serves as a flow control device which limits the total number of multipacket messages that can be handled by the subnet simultaneously. Therefore, removal of the throttling effect due to the small size PLT may introduce other congestion or stability problems. A corresponding consideration applies to the message number allocation.

The number of simultaneously unacknowledged 8-packet messages is further limited by the finite reassembly space in the destination IMP. In October 1974, a maximum of 34 buffers was available for reassembly (for IMP's without the VDH software). This meant that at most four 8-packet messages could be reassembled at the same time (leaving space for at least two single-packet messages). (The reassembly space must of course be shared with all other HOST's that are sending messages to the same destination IMP.) It may therefore become another serious throughput bottleneck.

From the above discussion, we know that even if there is no interference from other HOST's there cannot be more than four messages in transmission between any pair of HOST's due to the message number limitation. This restriction decreases the achievable throughput in the event that the line bandwidth times the round trip time is larger than four times the maximum message length. Fig. 2 depicts this situation. The input of the first packet of message i is initiated at time a after the last packet of message $i-1$ has been processed in the source IMP. After the input of this first packet is complete, the source IMP waits until time b when the RFNM for message $i-4$ arrives. Shortly after this RFNM has been processed (at time c) the transmission of the first packet over the first hop and

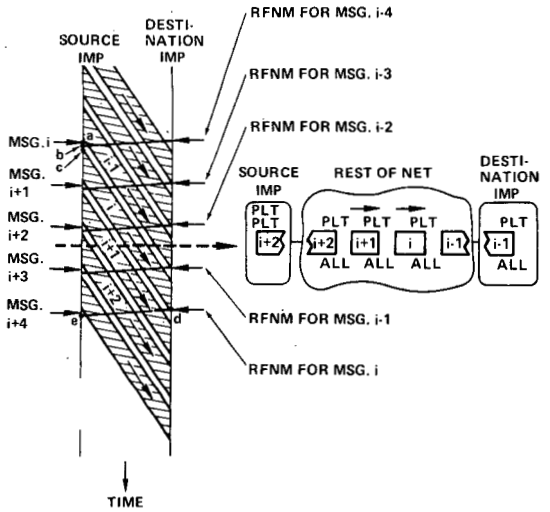


Fig. 2. The normal sequence of multipacket messages.

the input of the remaining packets from the HOST is initiated. At time *d*, all packets have been reassembled in the destination IMP, the first packet has been transmitted to the destination HOST and 8 reassembly buffers have been acquired by the RFNM which is then sent (with a piggy-backed ALL) to the source IMP. The RFNM reaches the source IMP at time *e* and thereby allows the transmission of message $i + 4$ to proceed. In this figure we also show a snapshot of the net at the time slice indicated by the dashed arrow. We show four messages (each with their own ALL and PLT): $i + 2$ is leaving the source IMP, both $i + 1$ and i are in flight, and $i - 1$ is entering the destination IMP. We also see the two unused PLT entries in the source IMP. The possible gaps in successive message transmissions represent a loss in throughput and can be caused by the limitation of four messages outstanding per IMP pair; this manifests itself in the next (fifth) message awaiting the return of a RFNM which releases one of the message numbers.

We have not yet mentioned the interference due to other store-and-forward packets which can significantly decrease the HOST-to-HOST throughput. This interference causes larger queueing times and possibly rejection by a neighbor IMP. Such a rejection occurs if either there are 20 store-and-forward packets in the neighbor IMP or if the output queue for the next hop is full. (There is an allowed maximum of 20 store-and-forward packets per IMP and of 8 packets for each output queue.) A rejected packet is retransmitted if no IMP-to-IMP acknowledgment has been received after a 125 ms timeout.

We now turn to a brief discussion of alternate routing and its impact on our throughput experiments. By alternate routing we refer to the possibility of sending data over two (or more) completely independent paths from source to destination. The shorter (shortest) path (in terms of number of hops) is usually called the primary path, and the longer path(s) are called secondary (tertiary, etc.) or alternate paths. For reliability reasons there should always be at least one alternate path available in a properly operating network. It turns out in the ARPANET that alternate paths are rarely used if they are longer than the primary path by more than two hops. The reason for this comes from the way the delay estimate is calculated, updated, and used by the routing procedure and from the way the output queues are managed. Each hop on the path from source to destination contributes four (arbitrary) units to the delay estimate. Each packet in an output queue between source and destination contributes one additional delay unit to the delay estimate. Since the length of the output queues is limited to 8 packets, one hop can therefore increase the delay estimate by at least 4, and at most, 12 units. Thus the minimum and maximum delay estimates over a path of n hops are, respectively, $4n$ and $12n$ delay units. Packets are always sent over the path with the smallest current delay estimate. From this, it follows that an alternate path is never used if it is more than three times longer (in terms of hops) than the primary path. Thus, for a primary path of length n, alternate routing is possible only over paths of length less than $3n$ hops. Let us assume that all the channels along the primary and alternate secondary path have the same capacity and that there is no interfering traffic. If we send as many packets as possible over the primary path, these packets usually will not encounter large queueing delays because this stream is fairly deterministic as it proceeds down the chain. This means that the delay estimate increases only slightly, although all of the bandwidth is used up. Therefore a switch to an alternate path occurs only if that path is slightly longer than the primary path. In the case of interfering traffic, the output queues will grow in size, and therefore a switch is more likely to occur. Such a switch to an alternate path may therefore help to regain some of the bandwidth that is lost to the interfering traffic. It has already been pointed out in [7] that, even if primary and secondary paths are equally long, at most a 30 percent increase in throughput can be achieved. This is due to the restriction of a maximum of 8 packets on an output queue and the fact that the frequency of switching between lines is limited to once every 640 ms (for heavily loaded 50-kbit/s lines). Thus the backlogged queue of 8 packets on the old path will provide overlapped transmission for only $8 \times 23.5 = 188$ ms of the total of 640 ms between updates (the only times when alternate paths may be selected). The relatively slow propagation of routing information further reduces the frequency of switching between the primary and secondary path. This discussion shows that alternate paths have only a small effect on the maximum throughput that can be achieved. However, the alternate paths are of great importance for the reliability of the network.

IV. THROUGHPUT EXPERIMENTS

The October 1974 throughput experiments produced the results shown in Fig. 3. Here we show the throughput (in kilobits/second) as a function of the number of hops between source and destination. Curve *A* is for the throughput averaged over the entire 10-min experiment; curve *B* is the throughput for the best block of 150 successive messages. Note that we are able to pump an average of roughly 37–38.5 kbit/s out to 5 hops[3]; it drops beyond that, falling to 30 kbit/s at 9 hops due largely to transmission gaps caused by the 4-message limitation. Also, the best 150-message throughput is not much

[3] This indicates an approximate efficiency of 75 percent on the 50-kbit/s lines. See [4] for a detailed description of line efficiency.

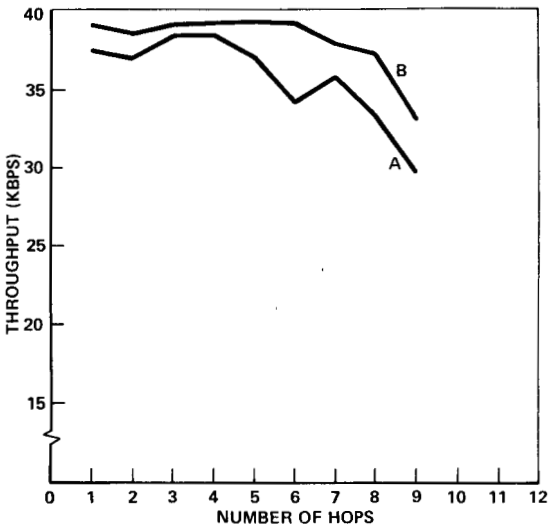


Fig. 3. ARPANET throughput (October 1974).

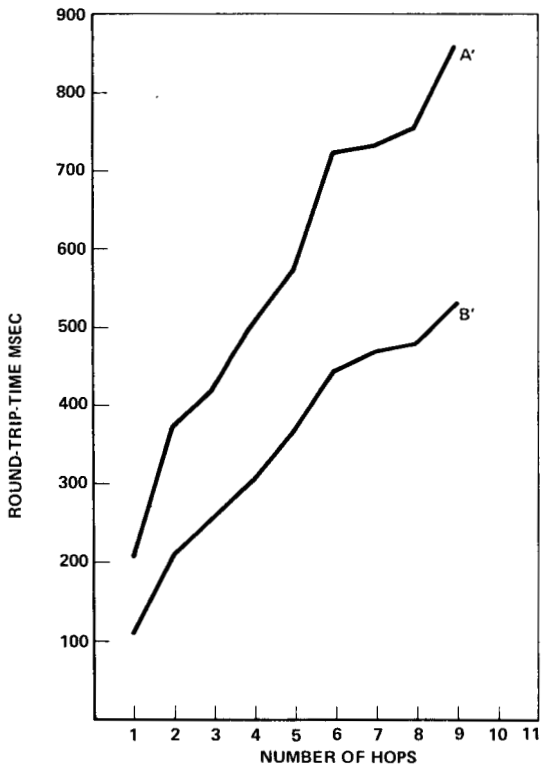


Fig. 4. Average round-trip delay in the ARPANET (October 1974).

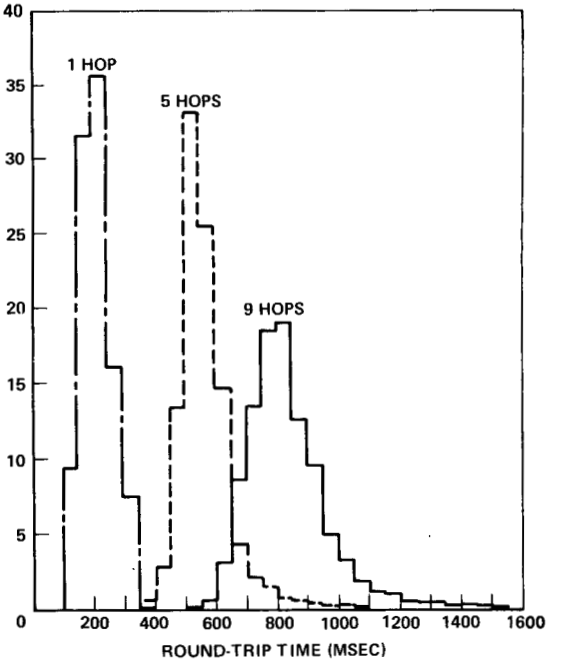


Fig. 5. Histogram of round-trip delay.

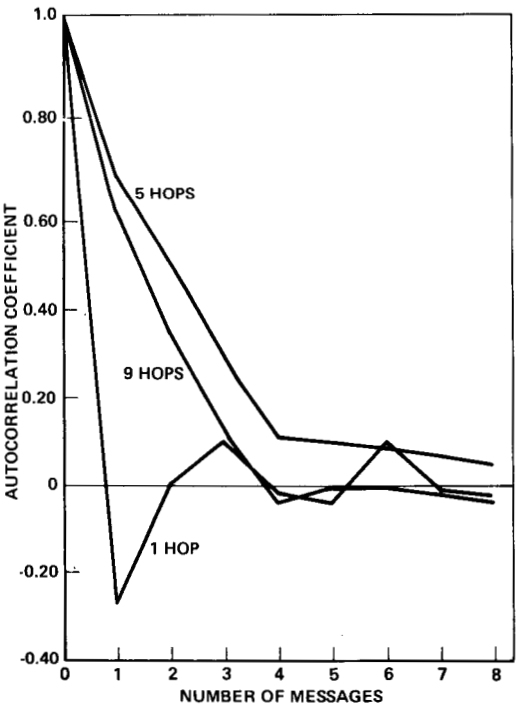


Fig. 6. Correlation coefficient for message delay.

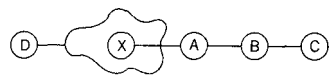
better than the overall average, indicating that we are almost achieving the maximal performance most of the time. In Fig. 4, we show the corresponding curves for the average round-trip delays (as seen by the UCLA PDP 11/45 HOST) as a function of source-destination hop distance; that is, curve A' is for the average and curve B' is for the best 150 successive messages. Note that the average delay for n hops may be approximated by $200 + 90(n - 1)$ ms. The measured histogram for delay is given in Fig. 5 for hop distances of 1, 5, and 9. Some of the large delays shown in this figure are caused by looping as explained in the next section. Of further interest is the autocorrelation coefficient of round-trip delay for successive messages in the network; this is shown in Fig. 6. Note that message delay is correlated out to about 3 or 4 successive messages.

V. LOOPING

The observation of occasional very long network delays recently led to an investigation of this phenomenon. The results showed that at times there was extensive looping in the subnet, i.e., packets were tossed back and forth between neighboring nodes many times and thus did not reach their destina-

tion until the loop was removed through the adaptive routing procedure. In what follows we describe how the ARPANET tries to avoid loops and why this procedure may fail in certain cases.

Let us consider a net with the following linear topology.



The exact topology between nodes D and A is immaterial for our discussion; node X is that node in the "rest of the network" which sends routing updates to node A. In this kind of configuration, nodes B and C should always send packets for node D to their left-hand neighbor (nodes A and B, respectively). We adopt the following notation to be used in the three following examples.

$B \rightarrow C$ means that node B sends a routing message to node C.
$d/1/A$ means that the overall delay estimate to node D is d units, the local delay over the best delay line to a neighbor is 1 unit, and A is the name of the best delay neighbor.

Table I describes an example of how loops can occur if no loop prevention procedure is used. The reader should review the earlier discussion which describes how delay estimates are formed.

Initially, the local delays in IMP's A, B, and C are zero, and the delay estimates to IMP D are, respectively, $d - 4$, d, and $d + 4$ delay units (row 0). Assume now that a sudden increase in traffic between node A and node D causes the delay estimate in A to be increased by 9 units (row 1). This fact is reported to B (row 2). Since C has not yet been informed of the sudden increase in traffic, it sends the old delay estimate to B. This causes B to consider C its best delay neighbor for IMP D (row 3); a loop between IMP's B and C has now been created! This loop remains effective until B tells C about the new situation (row 4), C reports back to B (row 7), and finally A's routing message causes B to switch back to A as its best delay neighbor (row 10). Since routing messages are sent every 640 ms, the loop persists for 640 to 1280 ms in this example.

To prevent the occurrence of this kind of loop in the ARPANET, a line hold-down mechanism was implemented [8]. The function of this mechanism was to continue using the best delay path for up to 2 s (ignoring the estimated delay from nonbest delay neighbors) whenever the delay estimate on this path increased by more than 8 delay units. The argument put forth in favor of this hold-down strategy was that, at times of sudden change, a node cannot be sure that its neighbors have already been informed of this change. Therefore, it should ignore the delay estimates from all but the best delay neighbor for some time until the information on the sudden change has propagated through the net.

Table II shows how this hold-down mechanism prevents the loop in the previous example. The hold-down of a line is indicated by an exclamation mark (!). Note that IMP's A and B start holding down their line to the best delay neighbor since their delay estimate gets worse by more than 8 delay units (rows 1 and 2). This causes IMP B to ignore the lower delay estimate received from IMP C and the loop is thereby prevented.

Since the decision of whether or not to initiate a line hold-down depends solely on the delay difference between consecutive routing messages, the hold-down mechanism is sensitive to the frequency at which routing messages are sent. The routing message frequency, however, is a function of line speed and line utilization. Therefore it is quite possible, for example, that A sends two routing messages to B before B sends one routing message to C. Table III gives an example of the occurrence of a loop which is due to the fact that routing messages on different lines are sent at different frequencies. In this case B does not initiate a line hold-down when it receives the routing information from A since the delay difference is always smaller than 8 (rows 2 and 4). Therefore, B switches the best delay path from A to C when it receives C's routing message (row 5), i.e., a loop has again been created! In row 6

TABLE I

	Routing	IMP A	IMP B	IMP C
0	initially	$d - 4/0/X$	$d/0/A$	$d + 4/0/B$
1	$X \rightarrow A$	$d + 5/4/X$		
2	$A \rightarrow B$		$d + 9/0/A$	
3	$C \rightarrow B$		$d + 8/0/C$	
4	$B \rightarrow C$			$d + 13/1/B$
5	$X \rightarrow A$	$d + 5/4/X$		
6	$A \rightarrow B$		$d + 8/0/C$	
7	$C \rightarrow B$		$d + 17/0/C$	
8	$B \rightarrow C$			$d + 21/0/C$
9	$X \rightarrow A$	$d + 5/4/X$		
10	$A \rightarrow B$		$d + 9/0/A$	

TABLE II

	Routing	IMP A	IMP B	IMP C
0	initially	$d - 4/0/X$	$d/0/A$	$d + 4/0/B$
1	$X \rightarrow A$	$d + 5/4/X$!		
2	$A \rightarrow B$		$d + 9/0/A$!	
3	$C \rightarrow B$		$d + 9/0/A$!	
4	$B \rightarrow C$			$d + 14/1/B$!
5	$X \rightarrow A$	$d + 5/4/X$!		
6	$A \rightarrow B$		$d + 9/0/A$!	
7	$C \rightarrow B$		$d + 9/0/A$!	
8	$B \rightarrow C$			$d + 13/0/B$!
9	$X \rightarrow A$	$d + 5/4/X$!		
10	$A \rightarrow B$		$d + 9/0/A$!	

TABLE III

	Routing	IMP A	IMP B	IMP C
0	initially	$d - 4/0/X$	$d/0/A$	$d + 4/0/B$
1	$X \rightarrow A$	$d + 1/2/X$		
2	$A \rightarrow B$		$d + 5/0/A$	
3	$X \rightarrow A$	$d + 5/4/X$		
4	$A \rightarrow B$		$d + 9/0/A$	
5	$C \rightarrow B$		$d + 8/0/C$	
6	$B \rightarrow C$			$d + 13/1/B$!
7	$X \rightarrow A$	$d + 5/4/X$		
8	$A \rightarrow B$		$d + 8/0/C$	
9	$X \rightarrow A$	$d + 5/4/X$		
10	$A \rightarrow B$		$d + 8/0/C$	
11	$C \rightarrow B$		$d + 17/0/C$!	

we see a hold-down at C. The situation becomes even worse when B receives the next routing message from C (row 11). Now B initiates a line hold-down in the wrong direction! This means that the B-C loop cannot be removed for almost 2 s because B ignores further delay estimates if received from A.

We call the occurrence of a loop whose existence is extended because of line hold-down a "*loop trap*." These loop traps have been observed repeatedly by the UCLA Network Measurement Center [9]. When such a loop trap occurs, packets are exchanged between neighbors up to 50 times before they can continue their travel to the destination IMP. We believe that these loop traps represent a major reason for the observation of occasional very long network delays during our throughput experiments.

Recently, the criterion for initiation of a hold-down was changed in such a way that it is now independent of the frequency at which routing messages are sent. As a result, we have not been able to detect loop traps in this modified system. Naylor has studied the problem of eliminating loops completely, and he presents a loop-free routing algorithm in [10].

VI. RECENT CHANGES TO MESSAGE PROCESSING

Some of the problems with the subnet control procedures described in Section III have recently led to a revision of message processing in the subnet.[4] In particular, message sequencing is now done on the basis of HOST-to-HOST pairs and the maximum number of messages that can be transmitted simultaneously in parallel between a pair of HOST's was increased from 4 to 8. Let us now describe the details of this new scheme.

Before a source HOST A at source IMP S can send a message to some destination HOST B at destination IMP D, a message control block must now be obtained in IMP S and IMP D. This message control block is used to control the transfer of messages. It is called a transmit block in IMP S and a receive block in IMP D. The creation of a transmit block-receive block pair is similar to establishing a (simplex) connection in the HOST-to-HOST protocol. It requires an exchange of subnet control messages that is always initiated by the source IMP. The message control blocks contain, among other things, the set of message numbers in use and the set of available message numbers.

After the first packet has been received from a HOST, the source IMP checks whether or not a transmit block-receive block pair exists for the transfer of messages from HOST A to HOST B. If HOST A has not sent any messages to HOST B for quite some time, it is likely that no such message control block pair exists. Therefore, source IMP S creates a transmit block and sends a subnet control message to destination IMP D to request the creation of a receive block. When IMP D receives this control message, it creates the matching receive block and returns a subnet control message to IMP S to report this fact. When IMP S receives this control message, the message control block pair is established.

[4] This is the "version 3" procedure in [5].

A shortage of transmit and/or receive blocks will normally cause only an initial setup delay. Currently, there are 64 transmit and 64 receive blocks available in each IMP. This means, for example, that a HOST can transmit data to 64 different HOST's simultaneously, or that two HOST's, attached to the same destination IMP, can each receive messages from 32 different HOST's simultaneously, etc. Since 64 message blocks is a rather large number, it is unlikely that this is a limiting resource.

The remaining resources are acquired in the following sequence: message number, reassembly buffers, and PLT entry. Since there are now 8 message sequence numbers which are allocated on a sending HOST-receiving HOST pair basis, a HOST is allowed to send up to 8 messages to some receiving HOST without having received an acknowledgment for the first message. For multipacket traffic this is more than enough because there are still only 6 entries in the PLT. Suddenly, therefore, the PLT has become a more prominent bottleneck than it used to be in the old message processing procedure when only four messages per IMP pair could exist.

Note that a multipacket message tries to obtain the reassembly buffers before it asks for the PLT entry. (This sequence for resource allocation can lead to difficulties as is described in Section VII.) In case there is no reassembly buffer allocation waiting at the source IMP, then as before, the message number and the PLT entry are used to send a REQALL to the destination IMP.

VII. THROUGHPUT FOR THE NEW MESSAGE PROCESSING

In February 1975, we repeated the throughput measurements of October 1974 to determine what effect the new message processing procedure had on the maximum throughput. Since the subnet had grown in size, we were able to measure the throughput as a function of hop distance up to 12 hops. The measured throughput in February 1975 with the new message processing procedure was significantly less than the throughput that was achieved in October 1974. For paths with many hops, the decrease in throughput was almost 50 percent. The observed throughput degradation was not due to a sudden surge of interfering traffic. Investigation of this performance degradation revealed the following two causes which explain in part the observed decrease in throughput: processing delays in 316 IMP's and an effect which we refer to as "phasing." Recent measurements show that the 316 IMP's in the ARPANET are becoming a major bottleneck. For the 316 IMP's, the queueing delays in the input or processing queue are, on the average, larger than the queueing delays in the output queue. The average queueing delay in the processing queue is about 10 ms. This is more than 5 times as much as the corresponding queueing delays in the 516 IMP. The cause for this increase in processing delay can be found in the more extensive processing which is done at a higher priority level.

A processing delay of 10 ms appears to be within acceptable limits. However, this is only an average number. In particular cases, we observed queueing delays in the input queue of several hundred milliseconds. In addition, it is not clear what

second-order effects these long processing delays have on a system that was originally designed to be limited by line bandwidth.

Phasing, the second (and more subtle) cause for the throughput degradation, is due to the sending of superfluous REQALL's! A REQALL is called superfluous if it is sent while a previous REQALL is still outstanding. This situation can arise if message i sends a REQALL but does not use the ALL returned by this REQALL because it obtained its reassembly buffer allocation piggy-backed on a RFNM for an earlier message (which reached the source IMP before its requested ALL). The sending of superfluous REQALL's is undesirable because it unnecessarily uses up resources. In particular, each REQALL claims one PLT entry. Intuitively, it appears to be impossible that more than four 8-packet messages could be outstanding at any time since there is reassembly buffer space for only four such messages (34 reassembly buffers). If, however, the buffer space that is freed when message i is reassembled causes an ALL to piggy-back on an RFNM of message $i-j$ ($j \geq 1$), then the RFNM for message i may queue up in the destination IMP behind $j-1$ other RFNM's! Thus only four messages really have buffer space allocated. In addition to these four, there are other outstanding messages which have already reached the destination IMP and which have RFNM's waiting for buffer space (i.e., waiting for piggy-backed ALL's).

The sending of more than four 8-packet messages is initially caused by the sending of superfluous REQALL's. The PLT entries which were obtained by these REQALL's are later used by regular messages. When the PLT is full, further input from the source HOST is stopped until a PLT entry becomes available (this results in the inefficient use of transmission facilities). Thus we have a situation where our HOST uses all six entries in the PLT for the transmission to a destination HOST.

Fig. 7 graphically depicts the kind of phasing we observed for almost all transmissions over more than 4 hops. Let us briefly explain the transmission of message i. At time a the last packet of message $i-1$ has been accepted and input of the first packet of message i is initiated. This first packet is received by the source IMP at time b. Since there is a buffer allocation available (which came in piggy-backed on the RFNM for message $i-7$) no REQALL is sent. However, the PLT is full at time b. Therefore, message i must wait until time c when the RFNM for message $i-6$ frees a PLT entry and message i may then proceed. At time d all 8 packets have been accepted by the source IMP. The first and eighth packet are received by the destination IMP at times e and f, respectively. The sending of the RFNM for message i is delayed until the RFNM's for messages $i-3$, $i-2$, and $i-1$ are sent. The buffer space that is freed when message $i+3$ reaches the destination at time g is piggy-backed on the RFNM for message i which reaches the source IMP at time h. This effect may be seen in Fig. 7 by observing the time slice picture while message i is in flight. Here we show messages as rectangles and RFNM's as ovals. Attached to RFNM's and messages are the ALL and PLT resources they own. We see the four ALL's owned by messages $i-1$, i, $i+1$ and by the RFNM for message $i-5$; we see the six PLT's owned by messages $i-1$, i and by the RFNM's for

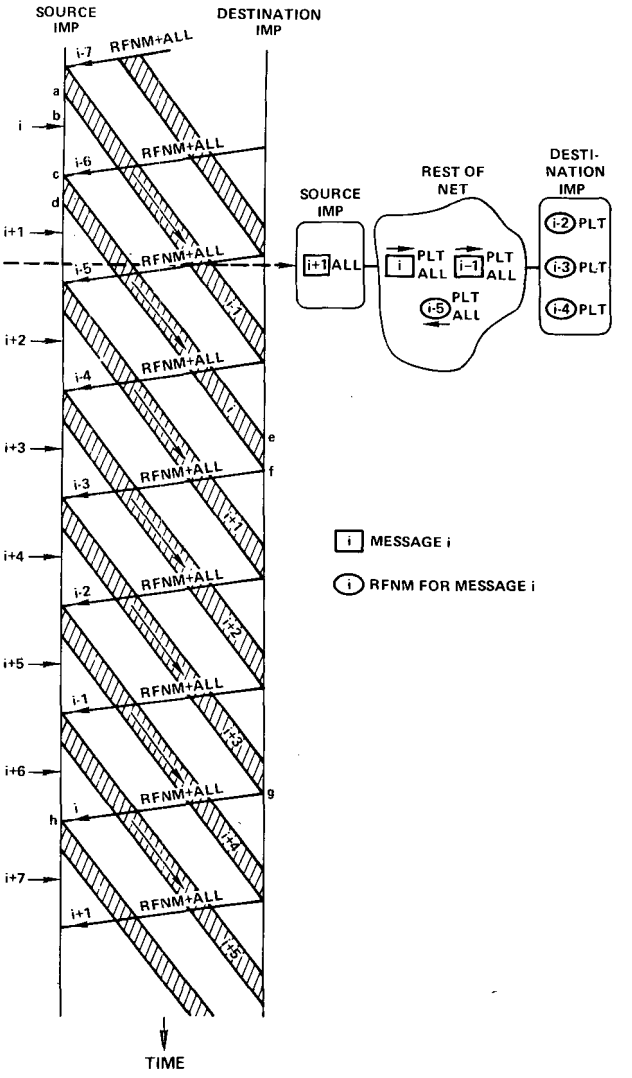


Fig. 7. Phasing and its degradation to throughput.

messages $i-5$, $i-4$, $i-3$, $i-2$. Message $i+1$ cannot leave the source IMP since it is missing a PLT; most of the PLT's are owned by RFNM's who are foolishly waiting for piggy-backed ALL's which are not critical resources at the source IMP (message $i+1$ has its ALL!). The trouble is clearly due to a poor phasing between PLT's and ALL's.

The phasing described above was observed for destination IMP's without the VDH software. For VDH IMP's, which can only reassemble one message at a time (10 reassembly buffers), a different kind of phasing was observed which resulted in even more serious throughput degradations! In this case, a situation is created in which a REQALL control message is sent for every data message. The 6 PLT's are assigned to 3 REQALL's and 3 data messages. Fig. 8 depicts this situation. The first packet of message i is transmitted from the source HOST to the source IMP between times a and b. Since there is no buffer allocation available, the source IMP decides to send a REQALL. However, all the PLT's are assigned and therefore the sending of the REQALL message is delayed until time c when the reply to an old REQALL (for message $i-3$) delivers an ALL and a PLT. At this time, the PLT entry is immediately stolen by the

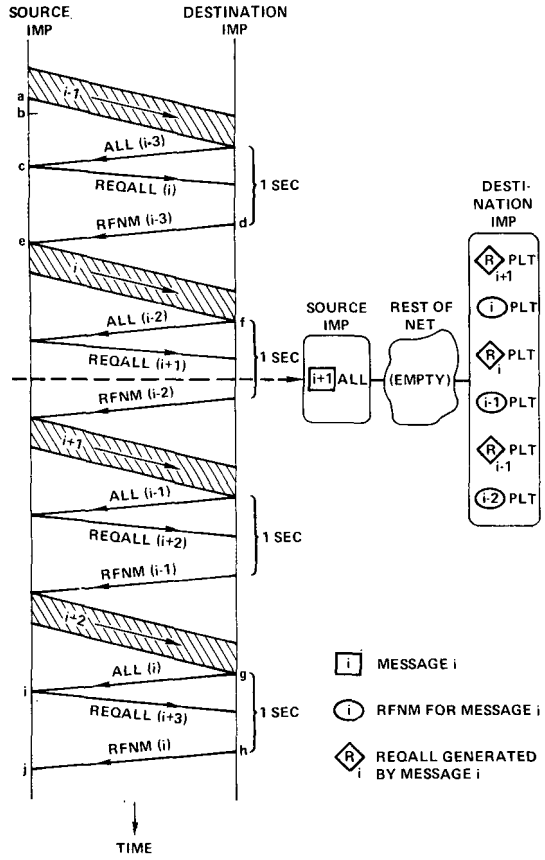


Fig. 8. Phasing when only one multipacket message can be assembled.

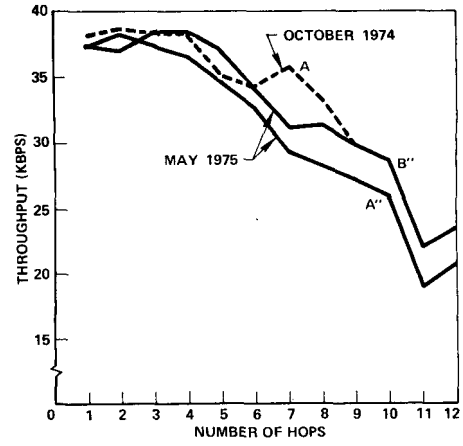


Fig. 9. ARPANET throughput (May 1975).

delayed REQALL (generated for message i). Note that at this point, message i gets the necessary buffer allocation but it cannot be sent to the destination because the PLT is once again full! Only when the RFNM for message $i-3$ times-out after 1 s (time d) and is received by the source IMP (time e) without a piggy-backed ALL does a PLT entry become free for use by message i. At time f, all 8 packets have been received by the destination IMP. The sending of the RFNM for message i is now delayed by several seconds because the replies for messages $i-2$, $i-1$ and for two previous REQALL's must be sent first (see the time-slice given by the dashed line in Fig. 8 which shows REQALL's as diamonds and is taken during the 1-s time-out when nothing is moving in the net). At time g, the ALL control message responding to REQALL (i) is sent to the source IMP, and 1 s later at time h the RFNM for message i times-out. The RFNM is finally received for message i by the source IMP at time j.

The phasing in the case of destination IMP's with VDH software results in throughput degradations by a factor of 3. This large decrease is due to the fact that the system is stalled for almost 1 s while the source IMP has the buffer allocation but no PLT entry; during this delay, the destination IMP, which can free a PLT entry by sending an RFNM, is waiting for the buffer allocation to use as a piggy-back. There are two obvious ways to avoid this undesirable phasing of messages. First, one can avoid sending superfluous REQALL's which are the underlying causes of the phasing. Secondly, one can avoid the piggy-backing of allocates on RFNM's as long as there are other replies to be sent. This second method was suggested and implemented by BBN.

In Fig. 9 we show some more recent throughput measurements made in May 1975 (after the phasing fix). As in Fig. 3, we show the throughput as a function of hop distance, with curve A'' displaying the throughput averaged over the 10-min experiment and curve B'' displaying the throughput for the best (maximum throughput) 150 consecutive messages. Curve A from Fig. 3 (October 1974) is included for a comparison of the two throughput experiments. We note that the throughput with the new message processing procedure is inferior to that in October 1974, although it is far better than that which we observed in February 1975 prior to the phasing fix.

VIII. CONCLUSIONS

In this paper we have described procedures for, limitations to, and measurement of throughput in the ARPANET. We identified some sources of throughput degradation due to the latest message processing procedure and displayed performance measurements after some of these problems were corrected. Here, as with many other deadlocks and degradations, it is rather easy to find solutions once the fault has been uncovered; the challenge is to identify and remove these problems at the design stage.

The ARPANET experience has shown that the building of a modern data communications network is an evolving process which requires careful observation and evaluation at each step along the way. Although the ARPANET was the first large-scale experimental packet-switched net and therefore underwent regular changes (as one would expect in any pioneering experiment) we foresee a continuing need for system evaluation.

The function of network measurements should not only be to test the initial configuration and make sure that it behaves according to specification. Indeed the rapid growth of these networks and the necessary changes in hardware and software make it extremely important to constantly reevaluate the total system design by means of analysis and measurements. This is the only guarantee for detecting performance problems as they arise and for acting accordingly before users experience degraded service.

REFERENCES

[1] G. Hicks and B. Wessler, "Measurement of HOST costs for transmitting network data," ARPA Network Information Center, Stanford Research Institute, Menlo Park, CA, Request for Comments 392, Sept. 1972.

[2] R. Kanodia, "Performance improvements in ARPANET teletransfer from multics," ARPA Network Information Center, Stanford Research Institute, Menlo Park, CA, Request for Comments 662, Nov. 1974.

[3] L. Kleinrock and W. E. Naylor, "On measured behavior of the ARPA network," in *AFIPS Conf. Proc.*, vol. 43, pp. 767-780, 1974.

[4] L. Kleinrock, W. E. Naylor, and H. Opderbeck, "A study of line overhead in the ARPANET," *Commun. Ass. Computing Machinery*, vol. 19, pp. 3-13, Jan. 1976.

[5] L. Kleinrock, *Queueing Systems, Vol. II: Computer Applications.* New York: Wiley, 1976.

[6] J. M. McQuillan, W. R. Crowther, B. P. Cosell, D. C. Walden, and R. E. Heart, "Improvements in the design and performance of the ARPA Network," in *AFIPS Conf. Proc.*, vol. 41, pp. 741-754, 1972.

[7] J. M. McQuillan, "Throughput in the ARPA network—Analysis and measurements," Bolt, Beranek and Newman, Inc., Cambridge, MA, Rep. 2491.

[8] —, "Adaptive routing algorithms for distributed computer networks," Rep. 2831, Bolt, Beranek and Newman, Inc., Cambridge, MA, 1974.

[9] W. E. Naylor, "A status report on the real-time speech transmission work at UCLA," Network Speech Compression Note 52, Dec. 1974.

[10] —, "A loop-free adaptive routing algorithm for packet switched networks," in *Proc. 4th Data Communications Symp.*, Quebec City, P.Q., Canada, Oct. 1975, pp. 6-1-6-11.

[11] H. Opderbeck and L. Kleinrock, "The influence of control procedures on the performance of packet-switched networks," *Nat. Telecommunications Conf.*, San Diego, CA, Dec. 1974.

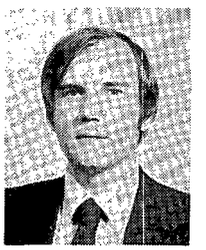
[12] B. Wessler, "Revelations in network HOST measurements," ARPA Network Information Center, Stanford Research Institute, Menlo Park, CA, Request for Comments 557, Aug. 1973.

[13] D. C. Wood, "Measurement of the user traffic characteristics," ARPA Network Information Center, Stanford Research Institute, Menlo Park, CA, Network Measurement Note 28, May 1975.

★

Leonard Kleinrock (S'55-M'64-SM'71-F'73), for a photograph and biography, see this issue, page 60.

★



Holger Opderbeck (S'73-M'74) received the B.S. and M.S. degrees from the University of Munich, Munich, Germany, in 1967 and 1969. In 1970 he was awarded a Fulbright scholarship for the study of computer science at the University of California at Los Angeles from which he received the M.S. and Ph.D. degrees in 1971 and 1973, respectively.

From 1969 to 1970 he worked as a Systems Analyst at Siemens, AG, Munich, Germany, where he did research in the area of information system design. From 1973 to 1975 he was Head of the ARPA Network Measurement Center at the University of California at Los Angeles. He is currently Director of Network Design with Telenet Communications Corporation, Washington, DC.

Dr. Opderbeck is a member of ACM.

Network Services in Systems Network Architecture

JAMES P. GRAY, MEMBER, IEEE

Abstract—**This paper discusses the services provided by a systems network architecture (SNA) network and design aspects related to these services. Both the basic transmission services and higher level network services are discussed.**

The first section describes the structure of SNA. The second section describes SNA's transmission services and sketches in the other aspects of SNA's structure. The next section describes services provided to users and managers of the network and the distribution of these services throughout the various nodes of the network. A concluding section discusses several potential extensions.

INTRODUCTION

SYSTEMS network architecture (SNA) is a system structure defined by message formats and protocols; it permits the design of products which can be connected together to form a unified communication-based data processing system. The architecture defines the appearance of each node in the network as seen by the network and the end users; that is, the external behavior of the network nodes is specified by SNA. Actual implementations realize this architected appearance in a variety of designs and utilize a variety of technologies. Familiarity with previous IBM communication products and software packages is assumed in explaining the reasoning behind SNA. References [1]-[13] contain other descriptions of SNA and implementations of SNA. SNA is an architecture; for details of implementation, including subsets of SNA that have been implemented, consult the product specifications.

SNA was developed to satisfy a specific set of requirements, the most important of which was the need to support distributed processing within a single application. This derived from a difficulty (communication facilities with reliability below that required of many major applications) and an opportunity (the price/performance improvements in microcoded controllers as a result of the successful development of LSI technologies). Since distributed processing implies the existence of distributed data and distributed application programs, this requirement became: develop a general solution for program to program communication through a network.

Manuscript received April 12, 1976; revised July 16, 1976.

The author is with the IBM Corporation, Research Triangle Park, NC 27709.